OSD Practical-8

Pre-lab:

1.Problems on address translation using paging, segmentation, and hybrid(both).

Ans:

Problem-01:

A certain computer system has the segmented paging architecture for virtual memory. The memory is byte addressable. Both virtual and physical address spaces contain 2^{16} bytes each. The virtual address space is divided into 8 non-overlapping equal size segments. The memory management unit (MMU) has a hardware segment table, each entry of which contains the physical address of the page table for the segment. Page tables are stored in the main memory and consists of 2 byte page table entries. What is the minimum page size in bytes so that the page table for a segment requires at most one page to store it?

Solution-

Given-

- Virtual Address Space = Process size = 2^{16} bytes
- Physical Address Space = Main Memory size = 2^{16} bytes
- Process is divided into 8 equal size segments
- Page table entry size = 2 bytes

Let page size = n bytes.

Now, since page table has to be stored into a single page, so we must have-

Size of page table <= Page size

<u>Size of Each Segment-</u>

Size of each segment

= Process size / Number of segments

= 2^{16} bytes / 8

= 2^{16} bytes / 2^3

= 2^{13} bytes

= 8 KB

<u>Number of Pages Of Each Segment-</u>

Number of pages each segment is divided

= Size of segment / Page size

= 8 KB / n bytes

= (8K / n) pages

Size of Each Page Table-

Size of each page table

= Number of entries in page table x Page table entry size

= Number of pages the segment is divided x 2 bytes

= (8K / n) x 2 bytes

= (16K / n) bytes

Page Size-

Substituting values in the above condition, we get-

(16K / n) bytes <= n bytes
(16K / n) <= n
n^2 >= 16K
n^2 >=214
n >= 2^7

Thus, minimum page size possible = 2^7 bytes = 128 bytes.

In-lab

1.Write a program to demonstrate Accessing Memory with Paging - linear translates.

Ans:

Vm.c code:

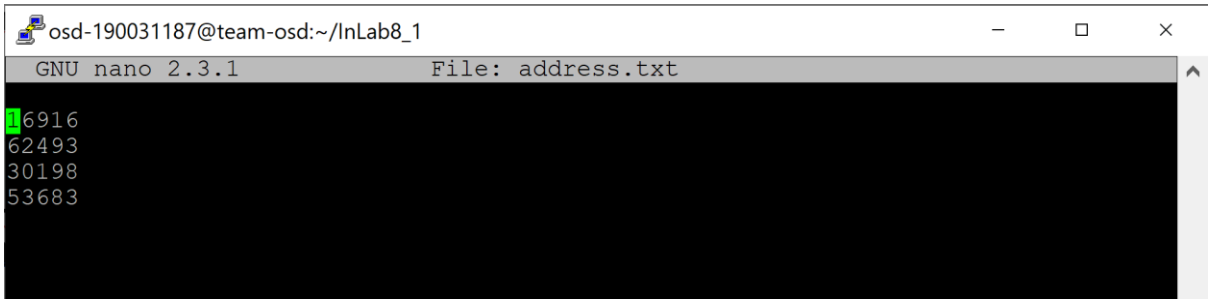
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#define PSIZE 4096
int main(int argc, char *argv[])
{
        int addressFile,backingStore;
        char *file= argv[1];
        char ch,ct, input[1000], output;
        int logicalAddress, physicalAddress;
        int i=0, j=0;
        int p,d;
        int f;
        char frames[PSIZE*16];
        int frametable[16];
        int start, current;
        int offset, pagefault=0;
        int freeFrame=-1;
        int pagetable[16];
        for (j=0;j<16;j++)
        {
                pagetable[j] = -1;
        }
    pagetable[0]=0x2;
    pagetable[1]=0x1;
    pagetable[2]=0x6;
    pagetable[3]=0x0;
    pagetable[4]=0x4;
    pagetable[5]=0x3;
    pagetable[9]=0x5;
    pagetable[11]=0x7;
        for (j=0;j<16;j++)
        {
                frametable[j] = -1;
        }
        frametable[0] = 1;
```

```
        frametable[1] = 1;
        frametable[2] = 1;
        frametable[3] = 1;
        frametable[4] = 1;
        frametable[5] = 1;
        frametable[9] = 1;
        frametable[11] = 1;
        addressFile = open("address.txt",O_RDONLY);
        backingStore = open("BACKING_STORE.bin",O_RDONLY);
        if(addressFile != -1)
        {
                while(read(addressFile, &ch, sizeof(char)) != 0)
                {
                        if(ch != '\n')
                        {
                                input[i] = ch;
                                i++;
                        }
                        else
                        {
                                logicalAddress =atoi(input);
                        p = (logicalAddress & 0x000000000000f000UL) >>  12;
                                d = (logicalAddress & 0x0000000000000fffUL);
                printf("\nlogicalAddress: %d, p: %d, d: %d", logicalAddress,p,d);
                                if(pagetable[p]!= -1){

                                        f = pagetable[p];
                                        physicalAddress = (f * PSIZE) + d;
                printf("\nphysicalAddress: %d, f: %d", physicalAddress,f);
                                }
//                              pagetable-miss, page-fault
                                else
                                {
                                        pagefault++;
//                                      locate free frame (-1) in physical memory
                                        for (j=0;j<8;j++)
                                        {
                                                if(frametable[j]==-1)
                                                {
                                                        freeFrame = j;
                                                        break;
                                                }
                                        }
                                        if(backingStore != -1)
                                        {
                                                offset=0;
                                                start = PSIZE * p;
```

```
                                current=lseek(backingStore, start, SEEK_SET);
                                        while((offset < PSIZE)&&(current))
                                        {
                                current = read(backingStore, &ct, sizeof(char));
                                        frames[freeFrame*offset] = ct;
                                                offset++;
                                        }
                                }
                                else
                                {
                                printf("Backing-Store Does not exist!");
                                        close(backingStore);
                                        close(addressFile);
                                        return 0;
                                }
//                              update pagetable, frametable
                                pagetable[p] = freeFrame;
                                frametable[freeFrame] = 0;
                                physicalAddress = (freeFrame * PSIZE) + d;
printf("\nphysicalAddress: %d, freeFrame: %d", physicalAddress, freeFrame);
                                }
                                output = frames[physicalAddress];
printf("\nByte value stored at physicalAddress %d: %c\n",physicalAddress,
output);
                                memset(input,0,sizeof(input));
                                i=0;
                        }
                }
                printf("\nTotal Page Faults: %d",pagefault);
        }
        else
                printf("Addresses File Does not exist!");
        close(backingStore);
        close(addressFile);
        return 0;
}
```

address.txt



pagetable.c

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

struct page{
	int page_no;
	int frame;
};
int main()
{
	int size_logical_address,size_physical_address,i,size_of_page,j;
	printf("Enter size of logical address space: ");
	scanf("%d",&size_logical_address);
	printf("Enter size of physical address space: ");
	scanf("%d",&size_physical_address);
	printf("Enter size of page: ");
	scanf("%d",&size_of_page);
	int number_of_frames = size_physical_address/size_of_page;
	int number_of_pages = size_logical_address/size_of_page;
	struct page pageTable[number_of_pages];
	printf("Enter page table: \n");
	for(i=0;i<number_of_pages;i++)
	{
		pageTable[i].frame = -1;
	}
	for(i=0;i<number_of_pages;i++)
	{
		int frame;
		bool replica = false;
		pageTable[i].page_no = i;
	printf("Enter frame for %d page number(-1 if frame doesn't exist): ",i);
		scanf("%d",&frame);
		for(j=0;j<number_of_pages;j++)
		{
			if(frame!= -1 && pageTable[j].frame == frame)
			{
				replica = true;
				printf("Frame number already stored\n");
			}
		}
		if(frame > number_of_frames)
		{
			replica = true;
			printf("Cannot exceed frame size\n");
```

```
                }
                if(replica == false)
                {
                        pageTable[i].frame = frame;
                }
        }
        int logical_address;
        printf("Enter -1 to exit\n");
        while(1)
        {
                printf("Enter logical address: ");
                scanf("%d",&logical_address);
                if(logical_address == -1)
                        return 0;
                int page_no = logical_address/size_of_page;
                int offset = logical_address%size_of_page;
                if(pageTable[page_no].frame == -1)
                {
                        printf("No such logical address exist\n");
                }
                else
                {
printf("Page no: %d \nOffset: %d\nFrame no: %d\nPhysical address:
%d\n",page_no, offset,
pageTable[page_no].frame,pageTable[page_no].frame*size_of_page + offset
);
                }
        }
}
```

BACKING_STORE.bin

```
osd-190031187@team-osd:~/InLab8_1
  GNU nano 2.3.1            File: BACKING_STORE.bin

                                                          !  "  #  $
%  &  '  (  )  *  +  ,  -  .  /  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?  @  A  B  C  D  E  F
G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e  f  g  h  i
j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~  €  ‚  ƒ  „  …  †  ‡  ˆ  ‰  Š  ‹  Œ  Ž  ‘
’  “  ”  •  –  —  ˜  ™  š  ›  œ  ž  Ÿ     ¡  ¢  £  ¤  ¥  ¦  §  ¨  ©  ª  «  ¬  -  ®  ¯  °  ±  ²  ³  ´  µ
¶  ·  ¸  ¹  º  »  ¼  ½  ¾  ¿  À  Á  Â  Ã  Ä  Å  Æ  Ç  È  É  Ê  Ë  Ì  Í  Î  Ï  Ð  Ñ  Ò  Ó  Ô  Õ  Ö  ×
Ø  Ù  Ú  Û  Ü  Ý  Þ  ß  à  á  â  ã  ä  å  æ  ç  è  é  ê  ë  ì  í  î  ï  ð  ñ  ò  ó  ô  õ  ö  ÷  ø  ù
ú  û  ü  ý  þ  ÿ

                                                          !  "  #  $  %  &  '  (  )  *  +  ,  -  .
/  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P
Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z  {  |  }  ~  €  ‚  ƒ  „  …  †  ‡  ˆ  ‰  Š  ‹  Œ  Ž  ‘  ’  “  ”  •  –  —  ˜  ™  š  ›
œ  ž  Ÿ     ¡  ¢  £  ¤  ¥  ¦  §  ¨  ©  ª  «  ¬  -  ®  ¯  °  ±  ²  ³  ´  µ  ¶  ·  ¸  ¹  º  »  ¼  ½  ¾  ¿
À  Á  Â  Ã  Ä  Å  Æ  Ç  È  É  Ê  Ë  Ì  Í  Î  Ï  Ð  Ñ  Ò  Ó  Ô  Õ  Ö  ×  Ø  Ù  Ú  Û  Ü  Ý  Þ  ß  à  á  â
ã  ä  å  æ  ç  è  é  ê  ë  ì  í  î  ï  ð  ñ  ò  ó  ô  õ  ö  ÷  ø  ù  ú  û  ü  ý  þ  ÿ

                              [ Read 1017 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Output

```
osd-190031187@team-osd:~/InLab8_1
[osd-190031187@team-osd InLab8_1]$ nano vm.c
[osd-190031187@team-osd InLab8_1]$ gcc vm.c
[osd-190031187@team-osd InLab8_1]$ ./a.out

logicalAddress: 16916, p: 4, d: 532
physicalAddress: 16916, f: 4
Byte value stored at physicalAddress 16916:

logicalAddress: 62493, p: 15, d: 1053
physicalAddress: 25629, freeFrame: 6
Byte value stored at physicalAddress 25629:

logicalAddress: 30198, p: 7, d: 1526
physicalAddress: 30198, freeFrame: 7
Byte value stored at physicalAddress 30198:

logicalAddress: 53683, p: 13, d: 435
physicalAddress: 29107, freeFrame: 7
Byte value stored at physicalAddress 29107:

Total Page Faults: 3
[osd-190031187@team-osd InLab8_1]$
```

2.Write a program that translates logical to physical addresses for a virtual address space of size 2^16 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB as well as a page table

Ans:

Virtual_mem.c code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <errno.h>
#define PAGE_SIZE 256
#define PAGE_ENTRIES 256
#define PAGE_NUM_BITS 8
#define FRAME_SIZE 256
#define FRAME_ENTRIES 256
#define MEM_SIZE (FRAME_SIZE * FRAME_ENTRIES)
#define TLB_ENTRIES 16
int
virtual,page_number,offset,physical,frame_number,value,page_table[PAGE_EN
TRIES],tlb[TLB_ENTRIES][2],tlb_front = -1,tlb_back = -1,mem_index =
0,fault_counter = 0,tlb_counter = 0;
char memory[MEM_SIZE];
int address_counter = 0;
float fault_rate,tlb_rate;
int get_physical(int virtual),get_offset(int virtual);
int get_page_number(int virtual);
void initialize_page_table(int n);
void initialize_tlb(int n);
int consult_page_table(int page_number);
int consult_tlb(int page_number);
void update_tlb(int page_number, int frame_number);
int get_frame();
int main(int argc, char *argv[]) {
   char* in_file;
   char* out_file;
   char* store_file;
   char* store_data;
   int store_fd;
   char line[8];
   FILE* in_ptr;
```

```
FILE* out_ptr;
initialize_page_table(-1);
initialize_tlb(-1);
if (argc != 4) {
   printf("Enter input, output, and store file names!");
   exit(EXIT_FAILURE);
}
else {
   in_file = argv[1];
   out_file = argv[2];
   store_file = argv[3];
   if ((in_ptr = fopen(in_file, "r")) == NULL) {
      printf("Input file could not be opened.\n");
      exit(EXIT_FAILURE);
   }
   if ((out_ptr = fopen(out_file, "a")) == NULL) {
      printf("Output file could not be opened.\n");
      exit(EXIT_FAILURE);
   }
   store_fd = open(store_file, O_RDONLY);
   store_data = mmap(0, MEM_SIZE, PROT_READ, MAP_SHARED, store_fd, 0);
   if (store_data == MAP_FAILED) {
      close(store_fd);
      printf("Error mmapping the backing store file!");
      exit(EXIT_FAILURE);
   }
   while (fgets(line, sizeof(line), in_ptr)) {
      virtual = atoi(line);
      address_counter++;
      page_number = get_page_number(virtual);
      offset = get_offset(virtual);
      frame_number = consult_tlb(page_number);
      if (frame_number != -1) {
         physical = frame_number + offset;
         value = memory[physical];
      }
      else {
         frame_number = consult_page_table(page_number);
         if (frame_number != -1) {
            physical = frame_number + offset;
            update_tlb(page_number, frame_number);
            value = memory[physical];
         }
         else {
            int page_address = page_number * PAGE_SIZE;
```

```
                if (mem_index != -1) {
                    memcpy(memory + mem_index,
                        store_data + page_address, PAGE_SIZE);
                    frame_number = mem_index;
                    physical = frame_number + offset;
                    value = memory[physical];
                    page_table[page_number] = mem_index;
                    update_tlb(page_number, frame_number);
                    if (mem_index < MEM_SIZE - FRAME_SIZE) {
                        mem_index += FRAME_SIZE;
                    }
                    else {
                        mem_index = -1;
                    }
                }
                else {
                }
            }
        }
        fprintf(out_ptr, "Virtual address: %d ", virtual);
        fprintf(out_ptr, "Physical address: %d ", physical);
        fprintf(out_ptr, "Value: %d\n", value);
    }
    fault_rate = (float) fault_counter / (float) address_counter;
    tlb_rate = (float) tlb_counter / (float) address_counter;
    fprintf(out_ptr, "Number of Translated Addresses = %d\n",
address_counter);
    fprintf(out_ptr, "Page Faults = %d\n", fault_counter);
    fprintf(out_ptr, "Page Fault Rate = %.3f\n", fault_rate);
    fprintf(out_ptr, "TLB Hits = %d\n", tlb_counter);
    fprintf(out_ptr, "TLB Hit Rate = %.3f\n", tlb_rate);
    fclose(in_ptr);
    fclose(out_ptr);
    close(store_fd);
  }

  return EXIT_SUCCESS;
}
int get_physical(int virtual) {
  physical = get_page_number(virtual) + get_offset(virtual);
  return physical;
}
int get_page_number(int virtual) {
  return (virtual >> PAGE_NUM_BITS);
}
```

```
int get_offset(int virtual) {
   int mask = 255;
   return virtual & mask;
}
void initialize_page_table(int n) {
   for (int i = 0; i < PAGE_ENTRIES; i++) {
      page_table[i] = n;
   }
}
void initialize_tlb(int n) {
   for (int i = 0; i < TLB_ENTRIES; i++) {
      tlb[i][0] = -1;
      tlb[i][1] = -1;
   }
}
int consult_page_table(int page_number) {
   if (page_table[page_number] == -1) {
      fault_counter++;
   }
   return page_table[page_number];
}
int consult_tlb(int page_number) {
   for (int i = 0; i < TLB_ENTRIES; i++) {
      if (tlb[i][0] == page_number) {
         tlb_counter++;
         return tlb[i][1];
      }
   }
   return -1;
}
void update_tlb(int page_number, int frame_number) {
   if (tlb_front == -1) {
      tlb_front = 0;
      tlb_back = 0;
      tlb[tlb_back][0] = page_number;
      tlb[tlb_back][1] = frame_number;
   }
   else {
      tlb_front = (tlb_front + 1) % TLB_ENTRIES;
      tlb_back = (tlb_back + 1) % TLB_ENTRIES;
      tlb[tlb_back][0] = page_number;
      tlb[tlb_back][1] = frame_number;
   }
   return;
}
```

addresses.txt

```
osd-190031187@team-osd:~/lnLab8_2
  GNU nano 2.3.1            File: addresses.txt

16916
62493
30198
53683
40185
28781
24462
54894
38929
32865
64243
2315
64454
55041
18633
14557
61006
62615
7591
                          [ Read 31 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

BANKING_STORE.bin

```
osd-190031187@team-osd:~/lnLab8_2
  GNU nano 2.3.1            File: BACKING_STORE.bin

                                                          !  "  #  $
%  &  '  (  )  *  +  ,  -  .  /  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?  @  A  B  C  D  E  F
G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e  f  g  h  i
j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~  €  ‚  ƒ  „  …  †  ‡  ˆ  ‰  Š  ‹  Œ  Ž  ‘
’  “  ”  •  –  —  ˜  ™  š  ›  œ  ž  Ÿ     ¡  ¢  £  ¤  ¥  ¦  §  ¨  ©  ª  «  ¬  -  ®  ¯  °  ±  ²  ³  ´  µ
¶  ·  ¸  ¹  º  »  ¼  ½  ¾  ¿  À  Á  Â  Ã  Ä  Å  Æ  Ç  È  É  Ê  Ë  Ì  Í  Î  Ï  Ð  Ñ  Ò  Ó  Ô  Õ  Ö  ×
Ø  Ù  Ú  Û  Ü  Ý  Þ  ß  à  á  â  ã  ä  å  æ  ç  è  é  ê  ë  ì  í  î  ï  ð  ñ  ò  ó  ô  õ  ö  ÷  ø  ù
ú  û  ü  ý  þ  ÿ

                                                   !  "  #  $  %  &  '  (  )  *  +  ,  -  .
/  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P
Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z  {  |  }  ~  €  ‚  ƒ  „  …  †  ‡  ˆ  ‰  Š  ‹  Œ  Ž  ‘  ’  “  ”  •  –  —  ˜  ™  š  ›
œ  ž  Ÿ     ¡  ¢  £  ¤  ¥  ¦  §  ¨  ©  ª  «  ¬  -  ®  ¯  °  ±  ²  ³  ´  µ  ¶  ·  ¸  ¹  º  »  ¼  ½  ¾  ¿
À  Á  Â  Ã  Ä  Å  Æ  Ç  È  É  Ê  Ë  Ì  Í  Î  Ï  Ð  Ñ  Ò  Ó  Ô  Õ  Ö  ×  Ø  Ù  Ú  Û  Ü  Ý  Þ  ß  à  á  â
ã  ä  å  æ  ç  è  é  ê  ë  ì  í  î  ï  ð  ñ  ò  ó  ô  õ  ö  ÷  ø  ù  ú  û  ü  ý  þ  ÿ

                          [ Read 1017 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

OUTPUT

osd-190031187@team-osd:~/InLab8_2

```
[osd-190031187@team-osd InLab8_2]$ nano virtual_mem.c
[osd-190031187@team-osd InLab8_2]$ gcc virtual_mem.c
[osd-190031187@team-osd InLab8_2]$ rm output.txt
[osd-190031187@team-osd InLab8_2]$ ./a.out addresses.txt output.txt BACKING_S
TORE.bin
[osd-190031187@team-osd InLab8_2]$ cat output.txt
Virtual address: 16916 Physical address: 20 Value: 32
Virtual address: 62493 Physical address: 285 Value: 32
Virtual address: 30198 Physical address: 758 Value: 32
Virtual address: 53683 Physical address: 947 Value: 32
Virtual address: 40185 Physical address: 1273 Value: 48
Virtual address: 28781 Physical address: 1389 Value: 32
Virtual address: 24462 Physical address: 1678 Value: 32
Virtual address: 54894 Physical address: 1902 Value: 32
Virtual address: 38929 Physical address: 2065 Value: 113
Virtual address: 32865 Physical address: 2401 Value: 40
Virtual address: 64243 Physical address: 2803 Value: 0
Virtual address: 2315 Physical address: 2827 Value: -89
Virtual address: 64454 Physical address: 3270 Value: 0
Virtual address: 55041 Physical address: 3329 Value: 60
Virtual address: 18633 Physical address: 3785 Value: -128
Virtual address: 14557 Physical address: 4061 Value: 81
Virtual address: 61006 Physical address: 4174 Value: 32
Virtual address: 62615 Physical address: 407 Value: 81
Virtual address: 7591 Physical address: 4519 Value: 32
Virtual address: 64747 Physical address: 4843 Value: 0
Virtual address: 6727 Physical address: 4935 Value: 32
Virtual address: 32315 Physical address: 5179 Value: 39
Virtual address: 60645 Physical address: 5605 Value: 32
Virtual address: 6308 Physical address: 5796 Value: 32
Virtual address: 45688 Physical address: 6008 Value: 32
Virtual address: 969 Physical address: 6345 Value: -30
Virtual address: 40891 Physical address: 6587 Value: 49
Virtual address: 49294 Physical address: 6798 Value: 32
Virtual address: 41118 Physical address: 7070 Value: -61
Virtual address: 21395 Physical address: 7315 Value: 32
Virtual address: 6091 Physical address: 7627 Value: -90
Number of Translated Addresses = 31
Page Faults = 30
Page Fault Rate = 0.968
TLB Hits = 1
TLB Hit Rate = 0.032
[osd-190031187@team-osd InLab8_2]$
```

POSTLAB

1.Write a program to demonstrate Accessing Memory with segmentation - linear translates.

Ans:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
intmain() {
     char name[100];
     char *description;
     strcpy(name, "Zara Ali");
     /* allocate memory dynamically */
     description = malloc( 200 * sizeof(char) );
   if( description == NULL ) {
   fprintf(stderr, "Error - unable to allocate required memory\n");
   }
    else {
    strcpy( description, "Zara ali a DPS student in class 10th");
    }
    printf("Name = %s\n", name );
printf("Description: %s\n", description );
}
```